Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem

Edmund Burke
University of Nottingham
University Park
Nottingham, UK
NG7 2RD
UK
ekb@cs.nott.ac.uk

Graham Kendall
University of Nottingham
University Park
Nottingham
NG7 2RD
UK
gxk@cs.nott.ac.uk

Abstract

This paper presents a new method to pack convex polygons into bins (the nesting problem). To do this polygons are placed in rows within bins using a meta-heuristic algorithm (simulated annealing) and by utilising the No Fit Polygon. We show that simulated annealing out performs hill climbing. The nesting algorithm is described in detail, along with various aspects that have been incorporated in order to improve the efficiency of the algorithm whilst maintaining solution quality. Computational results are given

1. Introduction

In the nesting problem pieces must be placed onto larger shape(s) (which we will refer to as stock sheets). In doing so two constraints must not be violated. The pieces must not overlap and all the pieces must lie within the confines of the stock sheet(s). In addition, it is usual to minimise the waste. The nesting problem only considers two dimensions, that is, the height and width of the stock sheet(s). Although it is possible to have a number of stock sheets, in this paper it is assumed that there is only a single stock sheet of fixed width and infinite height. This is common in the real world where a company may be cutting pieces from a roll of material. Of course, the roll does have a finite length but for the purposes of cutting the pieces it can be considered to be of infinite length.

One of the first references to the nesting problem is (Art, 1966). This paper also introduced the No Fit Polygon (NFP) (see section 2) which was also used by (Adamowicz, 1976) to calculate the minimum enclosing rectangle for two irregular shapes before placing the rectangle onto the stock sheet. It is the Adamowicz paper that is most often cited in the literature.

(Albano, 1980) used the no fit polygon in a heuristic search method to place irregular pieces onto stock sheets. The search is based on the A* algorithm but, because an admissible heuristic cannot be calculated, the search does not guarantee an optimal solution. In addition, other constraints are placed on the search due to the size of the search space. For example the size of the search tree is restricted. The evaluation function, f(n), is given by g(n) + h(n) where g(n) is the cost of the solution so far, which is a measure of the waste that cannot be used in later placements. h(n) is a heuristic measure that estimates the amount of waste in the optimal solution should the current piece be included in the placement. If it were possible to find suitable values for h(n) then optimal solutions could be found. (Oliveira, 1998) also used the no fit polygon to calculate the position of the next piece to be placed on the stock sheet. Once the best placement has been found the piece is added to the partial solution and the next piece is placed.

Utilising the no fit polygon makes it unnecessary to check for overlapping pieces. However, some researchers, while employing meta-heuristics, do allow overlaps during the search phase. (Oliveira, 1993) allows pieces to overlap and uses simulated annealing to move a single piece to another position. The objective function considers the length of the layout and combines this with a penalty term that penalises overlap. (Blazewicz, 1993) uses tabu search and an approach similar to (Oliveira, 1993) except that only non-overlapping pieces are moved.

Other researchers who have applied meta-heuristics approaches to the nesting (and related) problem include (Jain, 1992) who uses simulated annealing to position identical pieces together that could be “stamped out” from a length of material. A grid representation is used for the pieces which makes intersection very easy to compute. This approach is compared to (Jannsen, 1983) which splits the shapes into triangles and calculates the area of overlap using these triangles. (Cagan, 1994) used shape annealing which is a development of simulated annealing. Shapes are defined by grammars which dictate permissible orientations of the pieces. The shape annealing algorithm is used to determine whether a randomly selected shape rule should be applied to the current configuration. The example presented in the paper is to pack as many half hexagons into a predefined area. The results are acceptable but not provably optimal.

Heckmann and Lengauer (Heckmann, 1995) use simulated annealing, with a dynamic cooling schedule. As well as describing the cooling schedule and the type of moves allowed, the paper also discusses two different

ways that shapes can be represented. Two models were tested and the polygonal model was preferred to the raster model because of the improved accuracy given by this representation.
(Falkenauer, 1998), although not specifically addressing the nesting problem, considers Grouping Problems, using Genetic Algorithms (GA). The GA's are named Grouping Genetic Algorithms (GGA). One of the problems addressed is the bin packing problem. The nesting problem can be considered a bin packing problem where the cost associated with each piece is equal to its area.
The cost function used by Falkenauer is
Maximize

$$f_{BPP} = \frac{\sum_{i=1\ldots N}(F_i / C)^k}{N}$$

Where N is the number of bins used
F_i is the sum of the sizes of objects in bin i
C is the bin capacity
And k is a constant, k >1

This cost function measures the average 'bin efficiency' to the k^{th} power. Falkenauer achieved good results with k=2. Our tests have used both this cost function (suitably amended and detailed below) as well as a cost function that simply minimises the bin height.

This paper presents a method to produce solutions to the nesting problem. We employ a meta-heuristic technique (simulated annealing) and compare our results with hill climbing. The method utilises the no fit polygon as a way of packing the pieces.

2. No Fit Polygon

The No Fit Polygon (NFP) determines all arrangements that two arbitrary polygons may assume such that the shapes touch but so that they cannot be moved closer together without intersection. The example below shows how to calculate an NFP for two given polygons.

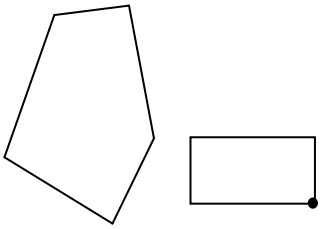

Figure 1 – Two Polygons

Consider the two polygons in figure 1. The aim is to find all arrangements such that the two polygons touch but do not intersect. If this can be achieved then we know that we cannot move the polygons closer together in order to obtain a tighter packing.

In order to find the placements we proceed as follows (see figure 2)

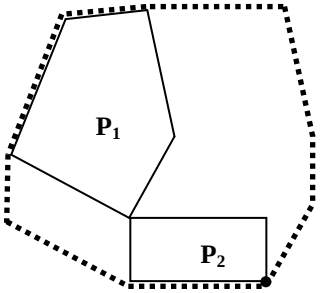

Figure 2 – NFP for Two Polygons

One of the polygons (P_1) remains stationery. The other polygon, P_2, obits around P_1, staying in contact with it but never intersecting it.
Both polygons retain their original orientation. That is, they never rotate.
As P_2 moves around P_1 one of its vertices (the reference point – shown as a filled circle) traces a line (this becomes the NFP).
This figure shows the starting (and finishing) positions of P_1 and P_2 with the NFP being shown as a dashed line. The NFP is slightly enlarged so that it is visible. In fact, some of the edges of the NFP will be identical to some of the edges of P_1 and P_2.
Once the NFP has been calculated for a given pair of polygons the reference point of P_2 can be placed anywhere on an edge or vertex of the NFP in the knowledge that P_2 will touch but not intersect P_1.

In order to implement an NFP algorithm it is not necessary to emulate one polygon orbiting another. (Cunninghame-Green, 1992) presents an algorithm that we use. It works on the assumption that (for convex polygons only) the NFP has its number of edges equal to the number of edges of P_1 and P_2. In addition, the edges of the NFP are simply copies of the edges of P_1 and P_2, suitably ordered. To build the NFP it is a matter of taking the edges of P_1 and P_2, sorting them and building the NFP using the ordered edges. This algorithm is also presented in (Cunninghame-Green, 1989), although here it is presented as a Configuration Space Obstacle (CSO).

3. Evaluation

3.1. Placing the polygons

In order to fill the bin (stock sheet) we proceed as follows. The first polygon to be placed is chosen and this becomes the stationery polygon (P_1 in the example above). The next polygon (P_2 from the above example) becomes the orbiting polygon. Using these two polygons the NFP is constructed.

The reference point of P_2 is now placed on each vertex of the NFP and for each position the convex hull for the two polygons is calculated. Once all placements have been considered the convex hull that has the minimum area is returned as the best packing of the two polygons. This larger polygon now becomes the stationery polygon and the next polygon is used as the orbiting polygon. This process is repeated until all polygons have been processed.

As each large polygon (i.e. output from the convex hull operation) is created, its width is checked. If this exceeds the width of the bin, then a new row within the bin is started. In this case the polygon which forced the width of the bin to be exceeded becomes the stationery polygon. That is, the large polygon built thus far forms one row and the next row is constructed using a single polygon as a starting point.

There are a number of points that need to be considered when using the above evaluation method. For reasons of space, a complete discussion of these points cannot be provided but they are fully described in (Burke, 1999). However the main points are summarised below

- A number of computational geometry algorithms have been implemented in order to effectively manipulate polygons. (Burke, 1999) details these algorithms.
- Executing computational geometry algorithms is relatively computationally expensive. This makes the evaluation function the slowest part of the algorithm. In view of this a cache has been implemented which retains *n* (user defined) previous evaluations. If a solution (partial or complete) is in the cache then the results are retrieved from the cache rather than executing the evaluation function. (Burke, 1999) explains how the cache has been implemented and shows how it significantly reduces the execution time of the algorithm.
- To further improve the algorithm the concept of a *polygon type* has been introduced. This allows polygons which are identical to be classified together so that the cache can work at a higher level of abstraction when deciding if a partial or complete solution is in the cache. (Burke, 1999) provides further details.
- Once the NFP has been calculated there may be a number of optimal placements for the two polygons being evaluated (that is, the convex hulls have equal area). In this case, the convex hull to use is selected at random. However, *all* the optimal solutions are stored in the cache so that if the solution (partial or complete) is seen again, one of the other convex hulls could be selected. (Burke, 1999) provides examples of this.
- If *n* polygons have been evaluated the solutions in the cache for the n^{th} polygon will be influenced by earlier evaluations. However, even though the convex hulls with the lowest area earlier in the evaluation has been chosen this may not be the best choice once other polygons are added to the solution. Therefore, it may be beneficial to re-evaluate a complete solution even if it is stored in the cache.
  In order to accommodate this we have introduced a *re-evaluation parameter* which, with some probability, forces a solution to be re-evaluated even if it is in the cache. (Burke, 1999) contains further details and shows that this parameter can be set to a low value.

3.2. Evaluating the placement

Two cost functions have been used in our experiments. The first simply minimises the height of the bin. The second method is based on the cost function used in (Falkenauer, 1998 – and shown above). Our cost function can be stated as

$$(\sum_{1}^{n}((1-(UsedRowArea/TotalRowArea))^{2})*k)/n$$

Where *n* is the number of rows in the bin
UsedRowArea is the total area of the polygons placed in the bin for that row
TotalRowArea is the total area of the stock sheet occupied by that row
k is a factor simply to scale the result. We used 100 but 1 could be used

In words, we are trying to minimise the area used by each row.

Both cost functions have been tested and the results are presented below.

4. Search Algorithms

As this is a new idea only using simulated annealing is being used at the present time (along with hill climbing). Simulated annealing (SA) (Kirkpatrick, 1983) is a search method based on hill climbing in that better states are always accepted. However, worse states are also accepted, with some probability. This allows the algorithm to jump out of local minima so that more of the search space can be explored. The probability that a worse state is accepted is given by

$P = \exp(-\Delta E/T)$

Where ΔE is the positive change in the energy level (the difference between the current evaluation and the evaluation of the neighbourhood state)
T is the current temperature of the system

We represent the cooling schedule as {*i, t, d, iter*}, where

- *i*, is the intial temperature
- *t*, is the terminating temperature
- *d*, is a formula used to decrement the temperature
- *iter*, defines how many iterations to carry out at each temperature

At present there are no known methods to calculate the cooling schedule for a large range of problems and the values are often set using empirical evidence from experimental runs of the algorithm. This is the approach we have adopted and we have also used the advice from (Rayward-Smith, 1996). Rayward-Smith states that you should start with a high temperature initially and monitor when between forty and sixty percent of worse states are being accepted. This is regarded as a good temperature for the initial value. We terminated the algorithm when the temperature reached zero. The other two parameters were set using experimentation.

Several neighbourhood functions have been implemented to allow us to explore the search space

Collect : Randomly selects a polygon and then scans through the remaining polygons and moves all polygons of the same type so that they are next to each other. The idea behind this function is that polygons of the same type will fit well together when packed.

NextDoor : Picks a polygon at random and swaps it with its next door neighbour. The motivation behind this function is to make small changes in the neighborhood in the hope that the search space will be systematically explored, leading to a good quality solution.

Random : Selects two polygons at random and swaps them

5. Testing and Results

5.1. Test Data

Two problems were used in our testing. The packing shown in figure 3 is from (Christofides, 1976). The reason this data was chosen is because it consists only of convex polygons and the optimum is known. The only change made is to multiply the measurements in the original paper by a factor of two. This was simply to assist us when displaying the results.

Our algorithm will not be able to find the optimum. The first row (A, B and C) can be constructed without problems, as can the second row (D, E, and F). However, to find the optimum for the third row, polygons would need to be presented in the order of G, H, I, J, K, L and M. The optimal solution could be built until the last polygon (M) came to be placed. At this time, due to the convex nature of the large polygon that had been built, the final polygon will not be placed in the position shown. In fact, the final polygon would be placed on a new row. Under these circumstances the total bin height would be 188 (as opposed to the optimal height of 140).

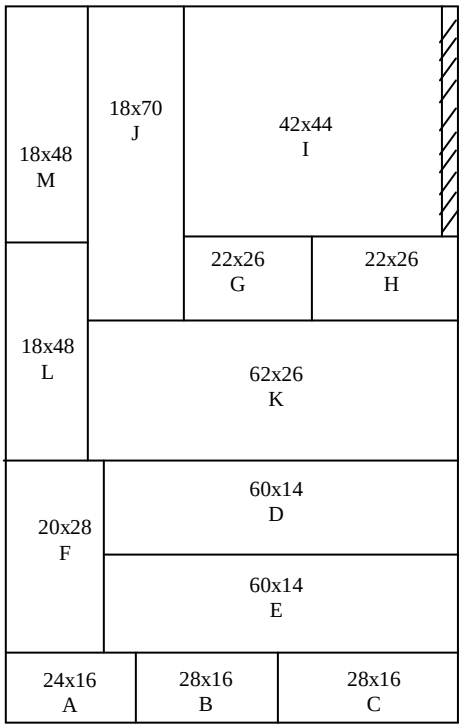

Figure 3 – Test Data 1

Therefore, using the permutation ABCDEFGHIJKLM produces a bin height of 188 (although, for the reasons described in section 3.1, it may take a few evaluations to achieve this). It appears to be a reasonable test to see if our algorithm can produce a bin height less than 188 as this would be a better solution that if we presented the shapes in optimal order, taking into account the convex constraints imposed by our method. We accept that we will never achieve a value of 140.

It should also be noted that several polygons are of the same type (BC, DE, GH, LM).

The second problem (figure 4) is taken from the real world. The objective is to cut polycarbonate shapes (used in constructing conservatories) from larger stock sheets.

In reality, the company receiving the orders has to cut many shapes from many stocks sheets whilst minimising the waste. The solution shown is one stock sheet on a given day. However, it is a worthwhile exercise to see how our algorithm packs these shapes.

Similar to the first problem, it is constrained by the convex properties of the algorithm. For example, the bottom four shapes cannot be packed in the way they have been as once three of the shapes have been packed,

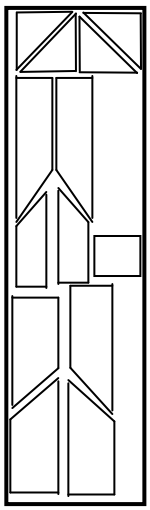

Figure 4 – Test Data 2

the fourth shape cannot be placed in the position shown.
The same is true for the four shapes above (excluding the rectangle).
The height of the stock sheet is 23940 units. Its width is 6240 units. Due to the convex properties of the algorithm it is not possible to achieve this solution but our main concern is to see if simulated annealing can perform better than hill climbing and also produce reasonable solutions.

5.2 Results

For both problems a number of experiments were conducted.

- A hill climbing algorithm.
- A simulated annealing algorithm using a linear cooling schedule of {*i*, 0, *NewTemperature* = *OldTemperature* – *n*, *iter*}
- A simulated annealing algorithm using a geometric cooling schedule of {*i*, 0, *NewTemperature* = *OldTemperature* * 0.9, *iter*}
- The neighbourhood functions described in section 4 were tested.
- Both cost functions described in section 3.2 were tested, although, where the bin height cost function is not used it is still shown in our results as it is an important measure of the quality of the solution.

All results are averaged over 15 runs of the algorithm.

Initially, the Random and NextDoor neighbourhood functions were tested against one another. The first set of test data and the modified Falkenauer cost function were used and the following results were obtained.

Function	Evaluation	Bin Height
Random	655.21	188.10
NextDoor	772.76	193.00

Table 1 – Random vs NextDoor Neighbourhood

Based on this result the Random neighbourhood function was used in preference to the NextDoor function.

An initial test comparing hill climbing against simulated annealing was also run. A very slow linear cooling schedule, {990, 0, temp = temp –15, 250}, and a hill climbing algorithm were run for the same number of iterations (16750) (and thus the same amount of time). This initial test was run on the second set of test data using the modified cost function of Falkenauer. The results were as follows.

Method	Evaluation	Bin Height
Simulated Annealing	1385.06	30312.40
Hill Climbing	1489.74	31605.60

Table 2 – Hill Climbing vs Simulated Annealing

This gave us an initial indication that simulated annealing would out perform hill climbing, although further results are presented below.

Following these tests we concentrated on the separate problems. Table 3 shows the results when the first set of test data was used. As well as comparing hill climbing against simulated annealing we are also comparing a linear and geometric cooling schedule, the two different evaluation functions and two of the neighbourhood functions.
From these results the following observations can be made

- Similar to the earlier tests, simulated annealing performs better than hill climbing.
- Using a geometric cooling schedule does lead to slightly better solutions but this is at the expense of run times that are about twice as long.
- Using the modified falkenauer cost function produces better solutions than using the bin height cost function although the difference is not as much as we had expected.
- The most significant result is that using the collect neighbourhood function produces better quality solutions than the random neighbourhood function, with reduced run times.

In conclusion we can say that the best quality results uses the collect neighbourhood function and uses simulated annealing with a cost function based on Falkenaur's. In addition, a linear cooling schedule can be used in order to produce faster results than a geometric cooling schedule.
Figure 5 show three of the best solutions found. The first figure shows a bin height of 158. The other two figures have a bin height of 162.
For the second test it was not possible to use the collect neighbourhood function as all the polygons have different dimensions (some may look the same but they are, in fact, all different). Therefore, we used the random function. The results produced are shown in table 4.

The best quality solutions were found using simulated annealing with a geometric cooling schedule but at the expense of a run time almost four times than that of the other methods. All simulated annealing algorithms again performed better then hill climbing but this time using the bin height cost function performed better than the modified falkenauer cost function.

6. Future Work

Our immediate aim is to compare simulated annealing with other meta-heuristic algorithms. Initially, we plan to develop a tabu search algorithm and also a genetic algorithm. We would also like to develop an ant algorithm for this problem.

In the longer term we plan to look at the following

- Allow rotation of polygons.
- Allow multiple bins (stock sheets).
- Consider ways of pre-processing the polygons so that some of the initial packing is done using a heuristic before the meta-heuristic algorithm runs.
- Allow for non-convex polygons. This, although increasing the complexity of the algorithm, should give better quality solutions.

7. Tables and Figures

Search Algorithm	Neighbour Function	Cost Function	Cooling Schedule	Eval	Bin Height	Time (Secs)
SA	Random	Modified Falkenauer	{40,0,t=t–1,50}	643.95	188.40	297
SA	Random	Bin Height	{40,0,t=t–1,50}		191.60	298
SA	Random	Modified Falkenauer	{40,0,t=t* 0.9,50}	705.70	189.20	643
HC	Random	Modified Falkenauer		752.64	196.40	300
SA	Collect	Modified Falkenauer	{40,0,t=t–1,50}	397.01	170.80	220
SA	Collect	Bin Height	{40,0,t=t–1,50}		169.07	221
SA	Collect	Modified Falkenauer	{40,0,t=t* 0.9,50}	390.05	168.13	431
HC	Collect	Modified Falkenauer		457.93	173.07	223

Table 3 – Hill Climbing vs Simulated Annealing over a variety of parameters

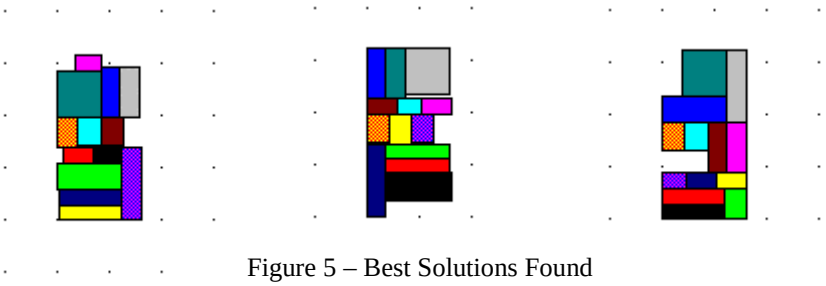

Figure 5 – Best Solutions Found

Search Algorithm	Neighbour Function	Cost Function	Cooling Schedule	Eval	Bin Height	Time (Secs)
SA	Random	Modified Falkenauer	{1000,0,t=t–25,50}	1661.77	33052.20	314
SA	Random	Bin Height	{1000,0,t=t–25,50}		31799.20	320
SA	Random	Modified Falkenauer	{1000,0,t=t*0.9,50}	1520.58	30999.60	1101
HC	Random	Modified Falkenauer		1683.53	33532.80	330

Table 4 – Results using Test Data 2

8. Conclusions

In this paper we have presented a new method for packing polygons onto stock sheets. This method uses the No Fit Polygon and lays the polygons in rows on a stock sheet. It has been shown that a meta-heuristic algorithm (simulated annealing) out performs hill climbing and produces good quality solutions. Various aspects of the problem have been tested. Using a linear cooling schedule performs almost as well as a geometric cooling schedule with the advantage of much reduced run times. Two different evaluation functions have been tested and both have shown to work, but on different problems. More work needs to be done in this area although we suspect that the modified falkenauer function will prove better when dealing with multiple stock sheets.

At present we are only working with convex polygons which means the optimal solution can never be found to some problems. However, we are able to find better solutions than if we packed the polygons using the permutation that would lead to the optimal solution if we allowed non-convex polygons. Now the method has been shown to be effective we plan to develop this work in a number of ways. For example, compare simulated annealing to other meta-heuristic algorithms and allow non-convex polygons as well as rotation.

9. References

- Albano, A., Sappupo, G. 1980. Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods. IEEE Trans. Syst., Man and Cybernetics, SMC-10, pp 242-248
- Art, R.C., 1966. An Approach to the Two-Dimensional Irregular Cutting Stock Problem. Technical Report 36.008, IBM Cambridge Centre.
- Adamowicz, M., Albano, A. 1976. Nesting Two-Dimensional Shapes in Rectangular Modules. Computer Aided Design, 8, 27-33
- Blazewicz, J., Hawryluk, P., Walkowiak, R. 1993. Using Tabu Search Approach for Soliving the Two-Dimensional Irregular Cutting Problem. Tabu Search (eds. Glover, F., Laguna, M., Taillard, E., Werra, D.), Vol. 41 of Annals of Operations Research, Baltzer, J. C. AG.
- Burke, E,K., Kendall, G. 1999. Implementation and Performance Improvement of the Evaluation of a Two Dimensional Bin Packing Problem using the No Fit Polygon. University of Nottingham Technical Report #ASAP99001.
- Cagan, J. 1994. Shape Annealing to the Constrained Geometric Knapsack Problem. Computer-Aided Design, 26, 10, 763-770
- Christofides, N., Whitlock, C. 1976. An Algorithm for Two_Dimensional Cutting Problems. Operations Research, 25, 30-44
- Cunninghame-Green, R. 1989. Geometry, Shoemaking and the Milk Tray Problem. New Scientist, 12, August 1989, 1677, pp 50-53.
- Cunninghame-Green, R., Davis, L.S. 1992. Cut Out Waste! O.R. Insight, Vol 5, iss 3, pp 4-7
- Falkenauer, E. 1998. Genetic Algorithms and Grouping Problems. John Wiley and Sons
- Heckmann, R., Lengauer, T., 1995. A Simulated Annealing Approach to the Nesting Problem in the Textile Manufacturing Industry. Annals of Operations Research, 57, 103-133
- Jain, P., Fenyes, P., Richter, R. 1992. Optimal Blank Nesting Using Simulated Annealing. Journal of Mechanical Design (Transactions of the ASME), March 1992, 114, 160-165
- Jannsen, T. L. 1983. A Simple Efficient Hidden Line Algorithm. Computers and Structures, Vol 17, No. 4, pp 563 - 571
- Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. 1983. Optimization by Simulated Annealing. Science, 220, 671-680
- Oliveira, J.F., Ferreira, J.S. 1993. Algorithms for Nesting Problems. In René Vidal, V.V, editor, Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems, pp 255-273, Springer-Verlag.
- Oliveira, J.F., Gomes, A.M., Ferreira, S. 1998. TOPOS A new constructive algorithm for nesting problems. Accepted for ORSpektrum
- Rayward-Smith, V.J., Osman, I.H., Reeves, C.R., Smith, G.D. 1996. Modern Heuristic Search Methods. John Wiley & Sons.